Pet Monitoring System

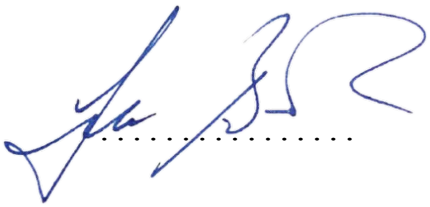
by
Can Karademir

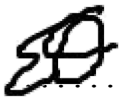
Engineering Project Report

Yeditepe University
Faculty of Engineering
Department of Computer Engineering
2025

Pet Monitoring System

APPROVED BY:

Prof. Dr. Şebnem Baydere ..... 
(Supervisor)

Dr. Öğr. Üyesi Esin Onbaşıoğlu ..... 

Dr. Öğr. Üyesi Tacha Serif ..... 

DATE OF APPROVAL: 15/06/2025

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Şebnem Baydere for her guidance and support throughout my project.

Also I would like to thank my parents for their support and encouragement throughout my education up to the present.

ABSTRACT

Pet Monitoring System

Pet owners often struggle to manage their pets' feeding schedules and toilet training due to busy daily routines. This project introduces the design of a smart pet monitoring system that enables remote access and behavior-based automation. The system consists of a Raspberry Pi-based smart feeding device, a Python-based server, and a Flutter mobile application. The device's camera provides live streaming, and the system uses artificial intelligence models to detect the animal's pooping behavior in real-time. When defecation is detected on a predefined training pad, a reward mechanism is triggered to reinforce proper habits. Users can also manually feed their pets through the mobile app. The system aims to improve toilet training efficiency and provide convenient remote monitoring and interaction for pet owners.

ÖZET

Evcil Hayvan İzleme Sistemi

Günümüzde artık hemen hemen her evde en az 1 evcil hayvan beslenmektedir ve insanların hayatlarında çok önemli noktalarda yer almaktadırlar. Tabi ki her zaman bu dostlarımızın yanında duramıyoruz ve evde yalnız bırakıyoruz. Bu yüzden evcil hayvanların uzaktan izlenmesi ve kontrolü, modern yaşam tarzı içerisinde artan bir ihtiyaç haline gelmiştir. Bu çalşmada, uzaktan evcil hayvanımızı besleme, canlı görüntüleme ve tuvalet eğitimini ödül temelli eğitim sağlayan bir ”Pet Monitoring System” geliştirilmiş olup. Sistem; Flutter tabanlo bir mobil uygulama, python ile yazılmış bir server, ve Raspberry Pi donanımı üzerine kurulu akıllı bir mama kabı ile çalışmaktadır. Cihaz üzerinde bulunan kamera ile canlı görüntü aktarımı ve sistem, köpeğin eğitim pedi üzerine dışkılama davranışını algılayarak, başarılı olan davranış üzerine ödül maması vererek köpeğin eğitimini destek olmaktadır. Kullanıcılar, mobil uygulama üzerinden manuel normal mama ve ödül maması verebilmekle birlikte canlı yayından ev ortamınını izleyebilmektedirler. Bu sistem, evcil hayvan sahiplerinin hem uzaktan etkileşim kurmalarına olanak tanımakta hem de hayvanların düzenli alışkanlıklar geliştirmesine katkıda bulunmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. Pet Monitoring System

Monitoring the behavior of pets and meeting their daily needs causes pet owners, especially those who have a busy life tempo, to stay out of the house for long periods of time during the day, which negatively affects the feeding and training of animals. In this context, the need for systems that work with remote access and offer automatic control is increasing.

In order to provide toilet training to pets, pet owners must be physically present in the same environment with their pets. However, thanks to developing image processing, IoT and mobile software technologies, this interaction can now be carried out remotely. In this context, the ”Pet Monitoring System” developed is designed as a system that can both detect a certain behavior of the dog (defecation) and reward it accordingly. The system enables the animal to be monitored with a live camera image, offers manual control to the user and contributes to animal training with an automatic reward mechanism based on behavior.

This study aims to reveal the design and applicability of real-time pet monitoring and interaction systems where software and hardware components work together.

Figure 1.1. Pet Training. Source: Retrieved from Google Images

1.2. Terms

The following terms are frequently used throughout this study and are defined to support better understanding of the system architecture and operation:

- *YOLO (You Only Look Once)* is a real-time object detection algorithm that performs classification and localization in a single step.
- *YOLO v11 Nano* is a lightweight version of the YOLO model optimized for low-power and embedded devices such as Raspberry Pi.
- *Keypoint Detection* is the method of identifying specific anatomical or structural points (e.g., joints) on a living being or object. It is used in this project to analyze the dog's posture.
- *Pooping Score* refers to a custom scoring mechanism based on keypoint distances and angles to estimate whether the animal is in a defecation posture.
- *Training Pad* is a predefined area on the floor used to train animals for proper defecation. The system only rewards the animal if the behavior occurs on this pad.
- *FastAPI* is a modern web framework for building APIs with Python. It is used in this project to manage user authentication and device communication.
- *Flutter* is an open-source UI toolkit developed by Google for building cross-platform applications from a single codebase. It is used for the mobile app in this system.
- *WebSocket* is a communication protocol that enables real-time, two-way communication between the server and clients. It ensures instant command exchange between the Raspberry Pi and the central server.
- *Servo Engine* is a small motor with precise control of angular position. In this project, it is used to release reward food by opening a mechanical lid.
- *Snapshot / Video Record* refers to media captured from the live stream of the Raspberry Pi camera. These can be triggered manually through the mobile application.
- *Inference* refers to the process of using a trained machine learning model to make predictions on new, unseen data. In this project, inference is performed on the Raspberry Pi using pre-trained YOLO and keypoint detection models to analyze the animal's behavior in real-time.

1.3. Motivation

Pets are not only animals but also family members for many people. However, today's busy work schedule and city life make it difficult for pet owners to take adequate care of their animals. Problems such as missing feeding times, disruption of toilet training and failure to observe behaviors can negatively affect the animal's habits and cause owners to feel guilty.

In this context, the need for systems that will allow remote monitoring of animals and automatic support of their training is increasing day by day. Existing commercial products usually only offer camera images and are limited to basic functions such as manual feeding. The absence of systems that offer features such as artificial intelligence-supported behavior analysis and reward-based toilet training has been the main source of motivation for this project.

This developed system allows pet owners to monitor their animals remotely, automatically detect behaviors such as defecation and reward correct behaviors. In this way, both the animal's habits are reinforced and the user is given the advantage of time and control.

1.4. Scope and Limitations

Scope:

- This project involves developing a smart system that can detect a dog's defecation behavior and automatically dispense food accordingly.
- The system consists of a Raspberry Pi device, camera, servo motor-based food bowl, a Python-based server, and a mobile application developed with Flutter.
- Image processing operations are performed using the YOLOv11 Nano object detection model and a keypoint detection model based on joint coordinates.
- The system's behavior analysis is carried out through a two-stage scoring process that combines both positional object detection and body posture estimation.
- The user can watch the live camera feed, manually feed the pet, capture photos and videos, and access recorded media through the mobile application.

Limitations:

- The limited processing power of the Raspberry Pi restricts the real-time use of more complex AI models.
- Detection performance may be affected in low-light or overly bright environments due to reduced image quality.
- The pet's distance from the camera may lead to inaccurate detection of keypoints.
- In rare cases, keypoint detection may fail for very furry or single-colored animals.
- The dataset used contains a limited number of examples, which reduces the model's generalization capability across all behavioral variations.
- The system requires an internet connection; remote features are unavailable during connection loss.
- Live stream interruptions may occur due to limitations of the free tunneling service used in the system.

1.5. Problem Definition

Toilet training of pets, especially during puppyhood, is a process that requires regular observation, timely rewards, patience, and continuity. However, due to today's living conditions, pet owners cannot always follow this process one-on-one, and the training becomes inefficient. Some pet owners are even required to pay significant amounts for professional trainers to assist with this process.

In addition, current automatic food dispenser systems only operate based on a timer and lack the ability to analyze the animal's behavior and respond to it accordingly. On the other hand, existing IP camera systems only provide basic visual monitoring functionalities.

The main problem addressed in this study is the lack of an intelligent system that can automatically detect the defecation behavior of pets and provide reward food accordingly. The proposed solution aims to both automate the training process and enhance the remote monitoring and control capabilities available to pet owners.

1.6. Requirements

• Hardware Requirements:

- Raspberry Pi 5 (4GB RAM or higher)
- Raspberry Pi Camera Module 3
- Servo Motor (PWM Controlled)
- 16GB+ microSD Card

• Software Requirements:

- Raspbian OS installed on the Raspberry Pi
- Python 3.9+ (for server-side and model execution)
- Flutter SDK (for mobile application development)
- OpenCV (for frame processing and camera handling)
- PyTorch (YOLOv11 Nano model inference)
- FastAPI (backend API and WebSocket communication)

• Network Requirements:

- Stable internet connection for the Raspberry Pi device
- Secure connection between mobile application and server (HTTP/HTTPS)
- (Optional) Remote access via port forwarding or tunneling service (e.g., Ngrok)

• User Requirements:

- Android or iOS device with the Flutter-based mobile application installed
- App-level permissions for camera, internet, and notifications
- Basic internet connection (for live monitoring and remote commands)

2. BACKGROUND

2.1. Previous Works

Remotely monitoring the behavior of pets and automatically responding to their actions has emerged as an important research domain with the advancement of computer vision and IoT technologies. While such monitoring was traditionally performed through direct physical observation by pet owners, modern technological solutions have made this process more effective and scalable. The proliferation of low-cost camera systems and lightweight artificial intelligence models has enabled real-time behavior analysis on embedded platforms [1].

Several studies in the literature focus on behavior monitoring for farm animals such as cattle or poultry. For instance, [2] proposed a YOLO-based architecture to classify behaviors like eating, lying, or walking in animals, while [3] introduced a system for multi-animal tracking and behavioral analysis based on pose estimation. These systems demonstrate that it is possible to quantify animal behavior using visual data and machine learning.

However, studies specifically targeting the detection of toilet-related behaviors and the integration of automated reward mechanisms for pets are extremely limited. This represents a significant gap in the current research landscape.

Some additional works [4] use object detection and keypoint estimation to classify general animal or human behaviors. Yet, the majority of these studies are performed on GPU-equipped desktop systems, which are not suitable for real-time embedded applications. Research on performing similar behavior analysis on resource-constrained platforms like the Raspberry Pi remains scarce.

This project utilizes both object detection and keypoint-based posture estimation to detect the defecation behavior of dogs in real time. When such behavior is observed, a reward mechanism is triggered automatically. In addition, the system offers real-time live streaming and interaction through a mobile application. With its integrated structure, this study makes a meaningful contribution by providing a low-cost, autonomous and intelligent pet monitoring system that works effectively on embedded hardware.

3. ANALYSIS

3.1. Problem Analysis

Training pets—particularly in developing consistent toilet habits—requires ongoing attention, patience, and timely reinforcement. However, many pet owners struggle to maintain this consistency due to demanding work schedules or long periods spent away from home. Although some commercial products offer features such as live video monitoring or timer-based food dispensers, these solutions lack behavioral awareness and real-time interactivity.

This project aims to bridge this gap by introducing a smart and responsive system that combines real-time behavior recognition with reward-based training. By automatically detecting defecation behavior and dispensing reward food when appropriate, the system helps reinforce desired actions—even in the owner's absence—making the training process more consistent, autonomous, and effective.

3.2. Requirements Analysis

This section outlines both the functional and non-functional requirements that the proposed Smart Pet Monitoring System must fulfill. Functional requirements describe the specific features and behaviors expected from the system, such as live streaming, reward dispensing, and behavior detection. Non-functional requirements, on the other hand, define performance-related expectations including latency, accuracy, reliability, and security. Together, these requirements form the foundation for system design, implementation, and validation processes.

Functional Requirements:

- View live stream from mobile application
- Trigger manual feeding (normal and reward)
- Start and stop real-time pooping detection
- Automatically dispense reward food upon detection
- Take snapshots and record videos
- Access recorded media gallery
- Manually mark training pad coordinates

Non-Functional Requirements:

- **Latency:** System must respond to pooping detections with reward delivery in under **1.5 seconds** (measured avg: **1.1s**).
- **Detection Accuracy:** The AI-based hybrid detection must maintain at least **95% F1 score** under optimal threshold settings.
- **Real-Time Performance:** Minimum **5 FPS** inference speed on Raspberry Pi 5 during active detection.
- **Uptime Reliability:** System must operate for **12+ hours continuously** without software crash or memory overflow.
- **Communication Security:** Data transfer between devices and server must be encrypted via **HTTPS or secure WebSocket**.
- **User Usability:** Mobile app must allow users to initiate any key action within **2 interactions (clicks or taps)**.

3.3. Use Case Diagram

The Use Case Diagram presented below outlines the primary interactions between the actors (User, Server, and Raspberry Pi) and the system functionalities. It represents how the user can perform tasks such as viewing the live stream, starting behavior detection, giving food manually, or accessing recorded media. The server acts as the coordinator that handles communication between the user and the smart device. The Raspberry Pi performs camera operations, runs AI models, and activates the servo motor, as illustrated in Figure 3.1.

3.4. Activity Diagram

The activity diagram below illustrates the flow of operations during the pooping detection and automatic reward process. It outlines how a command initiated by the user triggers a chain of actions involving AI-based behavior analysis, scoring, and physical reward dispensing through the Raspberry Pi device, as shown in Figure 3.2.

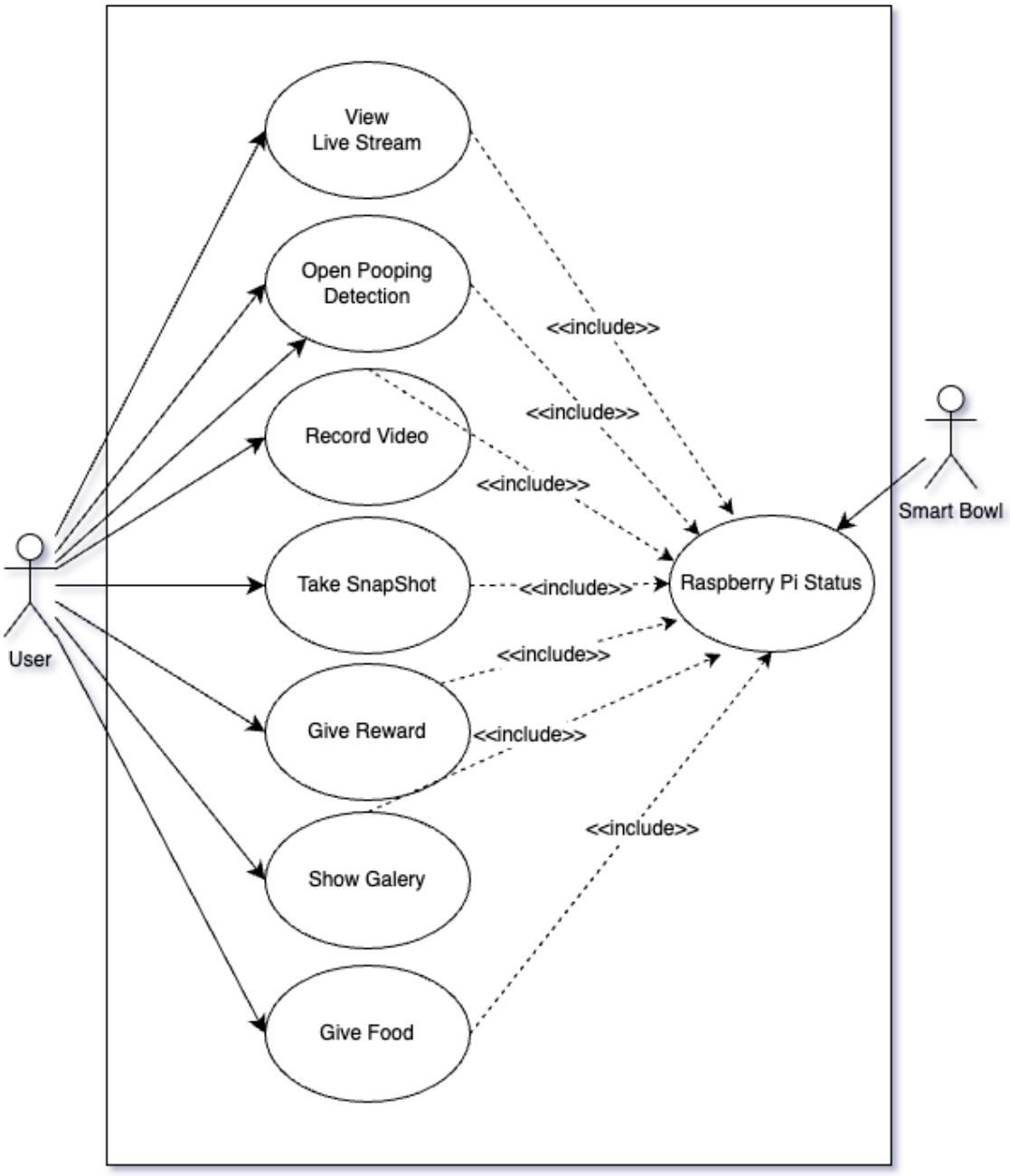


Figure 3.1. Use Case Diagram of the Pet Monitoring System

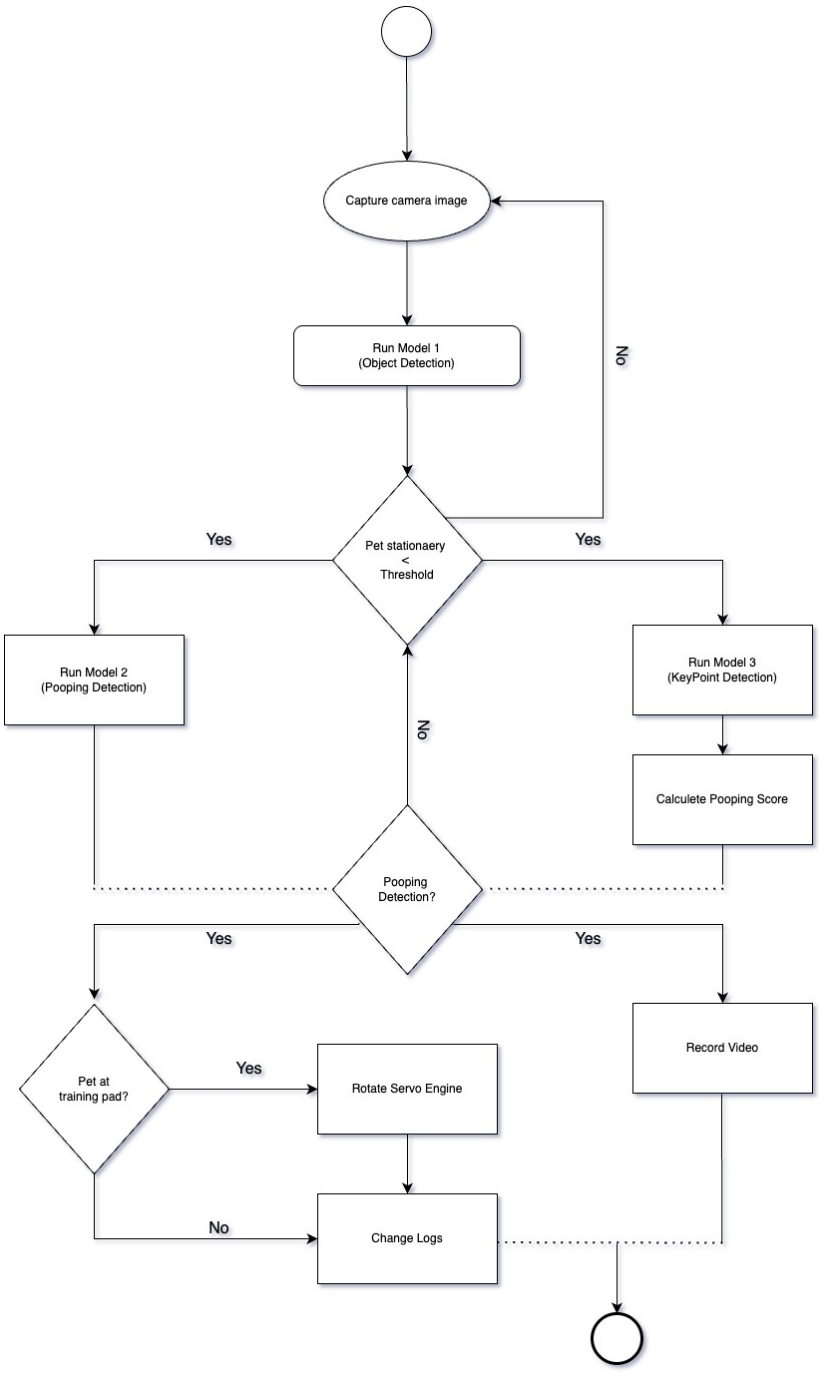


Figure 3.2. Activity Diagram of Pooping Detection and Reward Process

3.5. Sequence Diagram

3.5.1. Live Stream Initialization

The following sequence diagram demonstrates the communication flow that occurs when a user opens the live stream page in the mobile application. Upon entering the screen, the app sends an HTTP request to the FastAPI server to fetch the current stream URL of the selected Raspberry Pi device. The server responds with the dynamic IP address and stream path, which the app then uses to establish a direct MJPEG stream connection. Once connected, the app continuously fetches JPEG frames and renders them as real-time video, allowing the user to monitor the environment remotely, as illustrated in Figure 3.3.

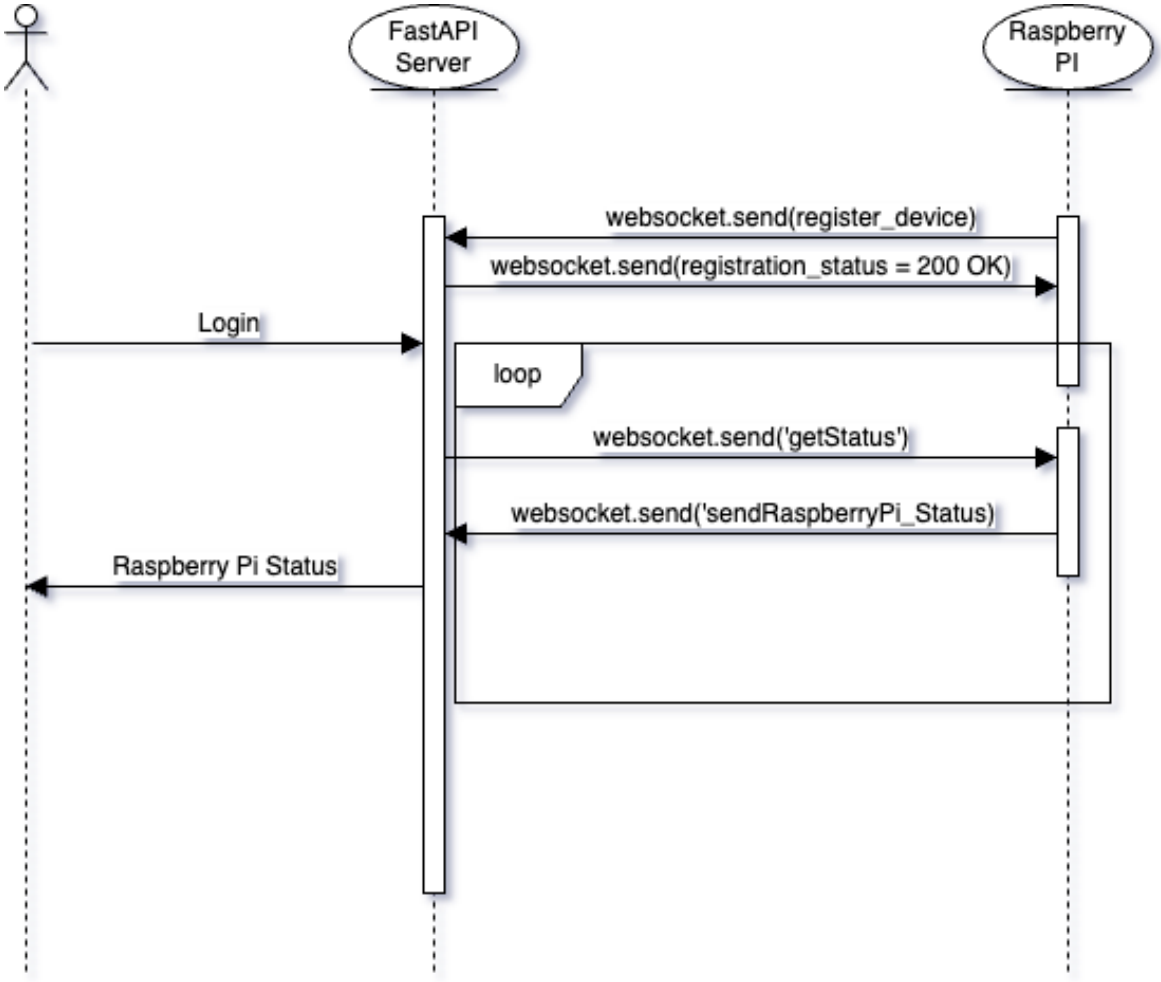


Figure 3.3. Sequence Diagram for Starting Live Stream

3.5.2. Manual Reward Feeding

The sequence diagram below illustrates the process that takes place when the user manually triggers the reward feeding function through the mobile application. Upon tapping the "Give Reward Food" button, the app sends an HTTP request to the FastAPI server. The server looks up the device's information and forwards the command to the Raspberry Pi over an active WebSocket connection. When command is received, the Raspberry Pi activates the servo motor to dispense reward food. After the operation completes, a confirmation message is sent back to the server and then forwarded to the mobile application to inform the user that the feeding was successful, as shown in Figure 3.4.

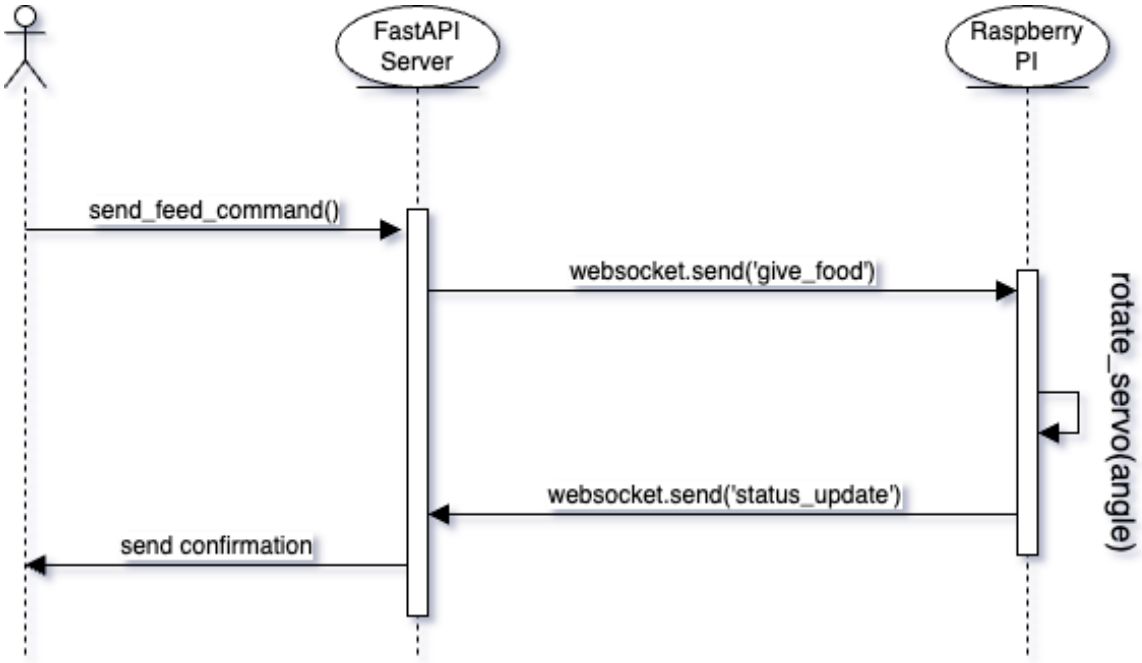


Figure 3.4. Sequence Diagram for Manual Reward Feeding

In this chapter, the system's functional structure and interaction logic have been analyzed in detail. The use case, activity, and sequence diagrams illustrate how users interact with the system and how core processes such as live streaming, pooping detection, and reward feeding are executed through coordinated communication between components. This analysis provides a solid foundation for the subsequent implementation and integration of each module.

4. DESIGN

4.1. General Structure of the System

The developed Pet Tracking System consists of a mobile application, a Python-based server, and a smart food bowl running on Raspberry Pi hardware. The system works with the cooperation of three basic components. The user can watch the live broadcast via the mobile application, send a manual food command, or start the behavior detection process. The commands sent are transmitted to the Raspberry Pi device via the server, and the ambient image is analyzed with the camera on the device.

Thanks to the YOLO-based models running in the device, the animal's behavior is classified in real time and certain behaviors such as "pooping" are detected. If this behavior occurs in the training pad area defined by the user, the system automatically gives a reward food. All these processes are carried out with a continuous data flow and control mechanism between the server and the device, as illustrated in Figure 4.1.

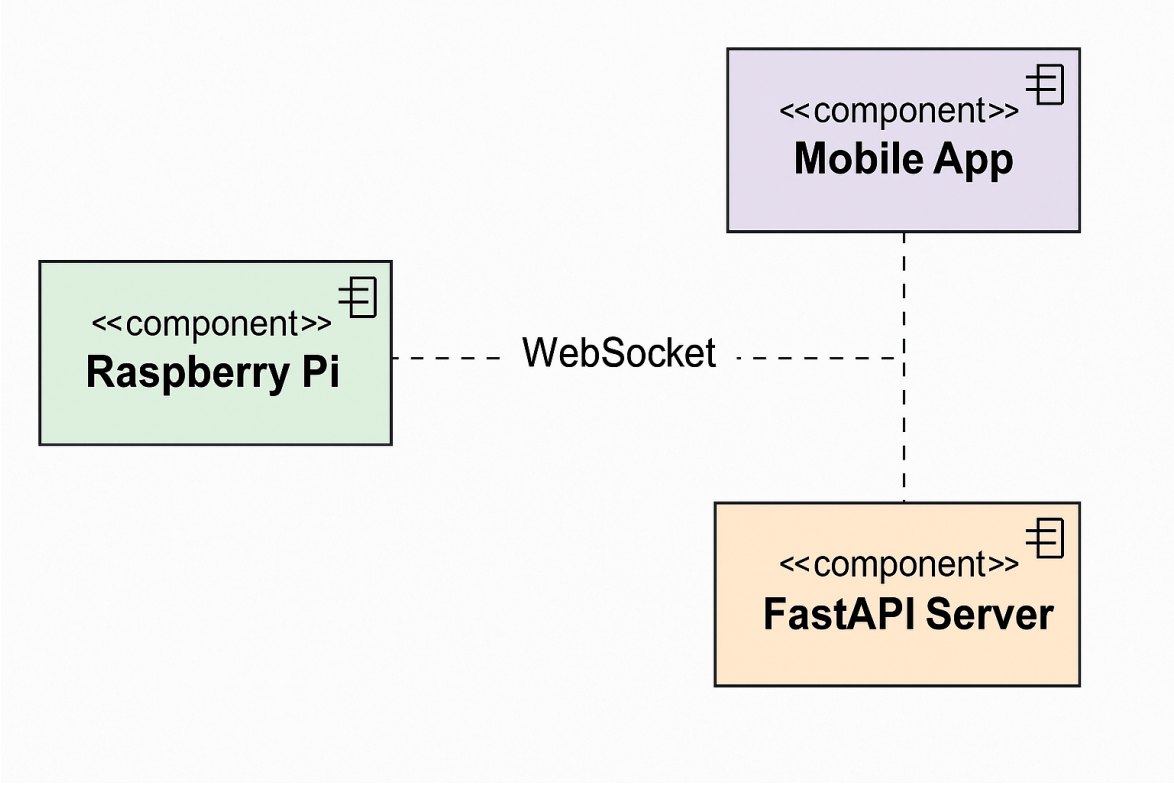


Figure 4.1. Component Diagram of the Pet Monitoring System.

4.1.1. Mobile Application

The mobile application offers a user-friendly interface developed with Flutter. Users can watch the live stream, manually give normal or reward food, and activate the pooping detection function. Snapshot and video captures are stored in the gallery and can be revisited. If the training pad is not automatically detected, users can manually mark its coordinates through the app. The main interface and gallery view of the application are shown in Figure 4.2. .

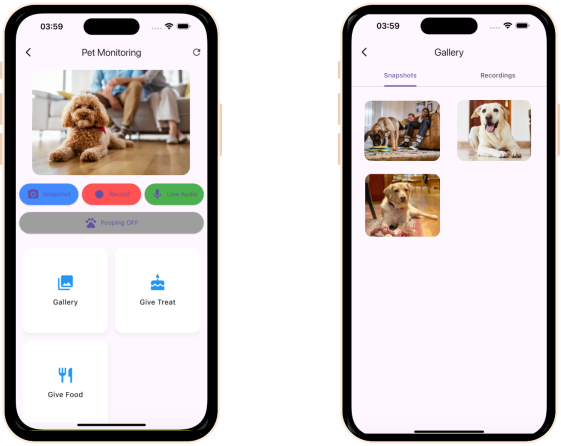


Figure 4.2. Mobile Application

4.1.2. FastAPI Server

The server, which is the central communication point of the system, was developed using the Python programming language and FastAPI, a modern web framework. The server directs the data exchange between the mobile application and Raspberry Pi devices, performs user and device authentication, and coordinates the entire command flow. Each Raspberry Pi device registers with the server with its deviceID and dynamic streamURL information during the initial connection. The server establishes a two-way and continuous connection with Raspberry Pi devices via

WebSocket connections; and receives commands from the mobile application via HTTP endpoints and directs them to the relevant device. At the same time, operations such as querying the current status of the devices, listing video and image recordings, or making them available for download are also performed through this server. Operations such as user management, device matching, command forwarding, and content flow are managed via this server at the center of the system, ensuring that all components work in harmony.

4.1.3. Raspberry Pi Based Smart Bowl

The Raspberry Pi 5 serves as the core embedded unit of the system, enabling real-time image processing and autonomous decision-making. Equipped with a high-resolution camera module and a servo motor, the device captures live environmental footage and physically delivers reward food when necessary.

The camera continuously streams video frames, which are analyzed locally using YOLO-based object detection and keypoint estimation models to detect the presence and behavior of the pet. If defecation is detected within the predefined training pad area, the system calculates a pooping score and triggers the servo motor to dispense a reward as positive reinforcement.

Thanks to its GPIO interface and modular design, Raspberry Pi interacts with both hardware components and the FastAPI-based server. This setup ensures synchronized operation across the device, server, and mobile application, allowing the system to operate in real time without external control. Physical views of the smart bowl used in the system are presented in Figure 4.3 and Figure 4.4.

Figure 4.3. Side View of The Smart Food Bowl.

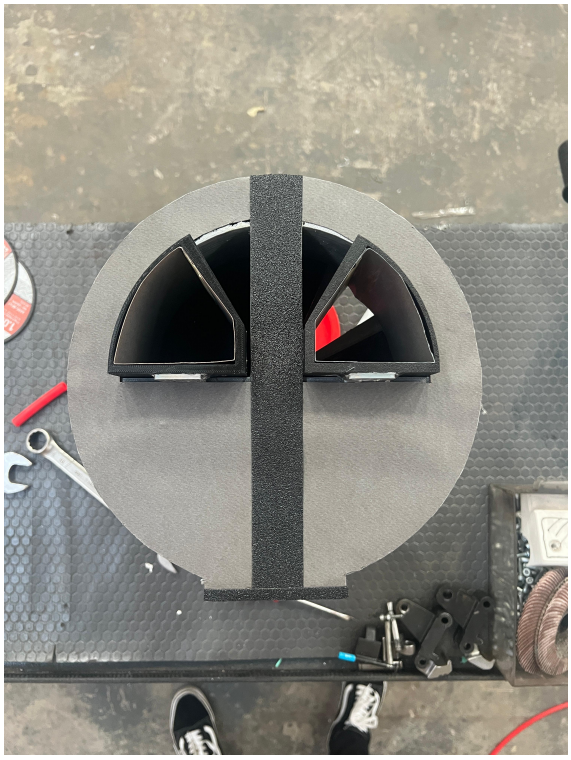


Figure 4.4. Top view of the smart food bowl.

The most important feature of the device is the artificial intelligence models that work directly on it. These models are three YOLO v11 Nano-based models configured to work in an optimized way on the Raspberry Pi device:

- **Model 1: General Object Detection** This model detects animals (cats or dogs) in the environment and determines their locations. It is also used in "stationary animal" detection to trigger other models of the system.
- **Model 2: Pooping Detection** This specially trained model divides the animal's behavior into four classes: Normal, Defecation, Scratching and Sitting. The model contributes to behavior analysis by making this classification in each frame.
- **Model 3: Keypoint Detection** This model, which is used to understand the animal's body position, detects 24 key points (paw, elbow, knee, tail, nose, etc.). A "pooping score" is calculated with the help of these points and the correctness of the behavior is supported by this score.

The outputs of the three models are combined to trigger the system's decision mechanism. If defecation is detected in the correct area, the servo motor is activated to deliver reward food as positive reinforcement.

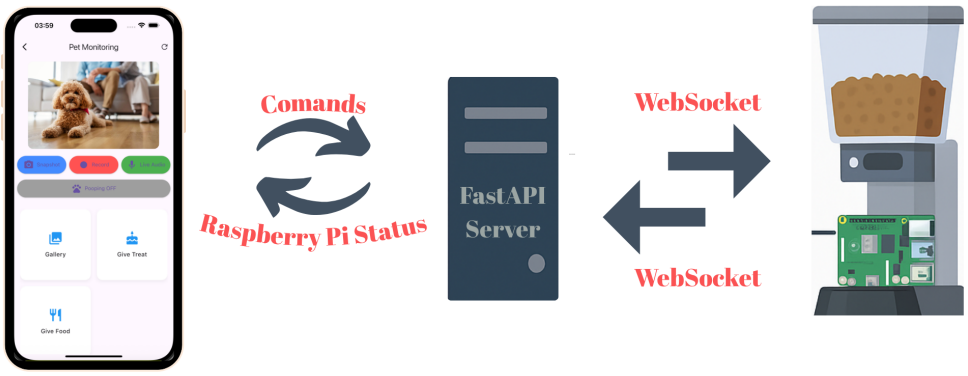


Figure 4.5. System Architecture of The Pet Monitoring System.

Additionally, the Raspberry Pi maintains a continuous WebSocket connection with the FastAPI server to receive commands and transmit data. This enables the device to operate both autonomously and in coordination with the central system, as illustrated in Figure 4.5.

4.2. Model 1: General Object Detection

4.2.1. Model Overview

Model 1 is responsible for detecting the presence of cats and dogs in the environment using the YOLOv11 Nano architecture. This model runs continuously on the Raspberry Pi and acts as a trigger for subsequent models when a motionless pet is detected within the camera frame.

The main purpose of this model is to provide fast and accurate object detection in a real-world environment. YOLOv11 Nano was chosen due to its lightweight structure, low inference latency, and compatibility with edge devices [5], [6].

In this system, when the model detects an animal and the animal remains motionless for a predefined period of time, it activates the behavioral analysis and keypoint estimation modules. Therefore, Model 1 acts as a gateway to initiate higher-level decision mechanisms.

4.2.2. Performance Evaluation

One of the important metrics used to ensure the accuracy of the model is the **F1 Score**. This metric evaluates the performance of the model by taking the harmonic average of the precision and recall values, representing the model's ability to correctly identify positive samples and not miss relevant instances.

The **precision value** of the YOLOv11 Nano model used in this system was measured as approximately 0.768 and the **recall value** as 0.695. The F1 score calculated based on these values is given by the following formula:

$$F1 = 2 \times \frac{P \times R}{P + R} = 2 \times \frac{0.768 \times 0.695}{0.768 + 0.695} \approx 0.73 \tag{4.1}$$

As a result, it is observed that the model performs in a balanced manner with high accuracy and recall capacity, making it suitable for real-time object detection tasks on edge devices.

4.3. Model 2: Pooping Detection

4.3.1. Model Overview

This model is a YOLOv11 Nano model trained to distinguish pet pooping behavior from other daily positions [2], [4]. It is one of the three main AI models in the system and is triggered when Model 1 detects the pet as motionless in the camera frame. The model runs on a separate thread and classifies the pet's posture into four specific behavior classes:

- Normal
- Pooping
- Scratching
- Sitting

The model is optimized for real-time use on low-powered embedded systems like Raspberry Pi. YOLOv11 Nano was chosen due to its lightweight architecture, fast inference speed, and suitability for real-time decision-making. The main objective of this model is to detect the defecation behavior of pets with high classification accuracy by evaluating each frame individually.

4.3.2. Dataset and Training Setup

The dataset for this model was curated and annotated by the author using the Roboflow platform, and is hosted at [7]. It includes 5969 labeled images representing various pooping-related behaviors of dogs and cats in indoor environments.

Data Separation:

Training Set	Validation Set	Test Set
2331 images	292 images	291 images

Preprocessing: All images were resized to 640×640 px and auto-oriented for consistent input.

Data Augmentation: To enhance generalization and reduce overfitting, each image was augmented with the following techniques:

- 3 synthetic samples per original image
- Rotation: ±15°
- Shear: ±10° horizontal and vertical
- Brightness: Random variation up to ±15%

Training Configuration:

4.3.3. Training Configuration

- Platform: Google Colab with GPU
- Model: YOLOv11 Nano
- Epochs: 100
- Batch Size: 8
- Optimizer: Adam
- Loss Functions: box_loss, cls_loss, dfl_loss

Public Dataset Link:

A publicly available dataset titled *Pooping Detection* was used for training purposes[1] . This dataset and its configuration enabled the model to effectively learn pet behavior and achieve high classification performance across various test environments.

This dataset and configuration enabled the model to learn pet behavior efficiently and achieve high classification performance across multiple test environments.

4.3.4. Performance Evaluation

To assess the model's performance in recognizing pooping-related behaviors under real-time conditions, standard evaluation metrics such as **precision**, **recall**, **F1 score**, and **mean Average Precision (mAP)** were employed. These metrics offer a clear and comprehensive view of the model's classification capabilities in practical scenarios.

[1] `https://universe.roboflow.com/test-ja6hg/pooping-detection`

Results:

Precision: 0.927 **Recall:** 0.941

mAP@0.5: 0.958 **mAP@0.5:0.95:** 0.835

F1 Score Calculation:

$$F1 = 2 \times \frac{P \times R}{P + R} = 2 \times \frac{0.951 \times 0.958}{0.951 + 0.958} \approx 0.955 \tag{4.2}$$

This balanced F1 score confirms that the model successfully identifies pooping behaviors while maintaining a low false positive rate.

Evaluation Graphs

The Pooping Detection model's training performance was evaluated using key metrics. The graphs below show how classification accuracy improved:

- **F1 Score:** Harmonic mean of precision and recall. A rising trend means better balance, as shown in Figure 4.6.
- **Precision:** Ratio of correct positive predictions. Higher values indicate fewer false positives, as shown in Figure 4.7.

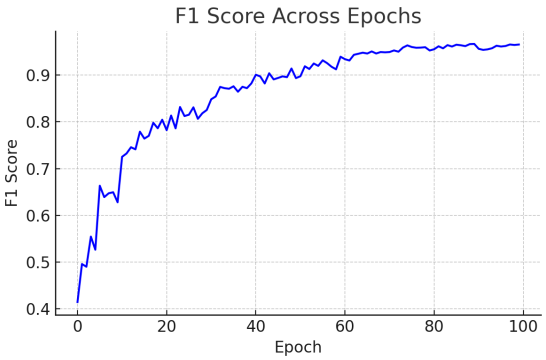


Figure 4.6. F1 Score of Model 2

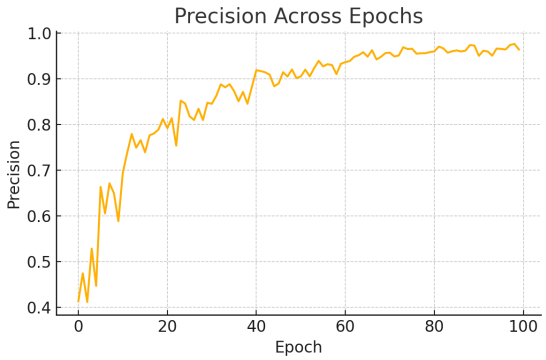


Figure 4.7. Precision of Model 2

Additionally, the following metrics reflect how well the model identifies relevant instances and generalizes to unseen data:

- **Recall:** Ratio of correctly predicted positives among all actual positive samples. High values indicate that the model effectively detects pooping-related poses, as shown in Figure 4.8.
- **mAP@0.5 and mAP@0.5:0.95:** Measure localization and classification performance across varying IoU thresholds. These scores indicate robustness and accuracy, as illustrated in Figure 4.9.

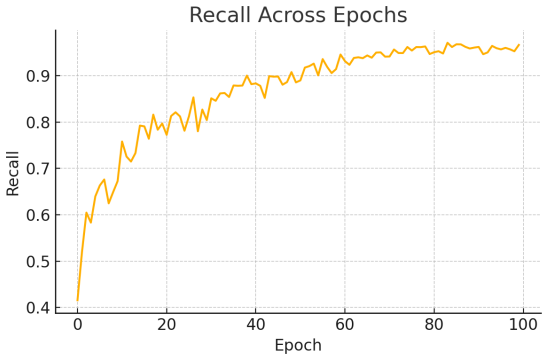
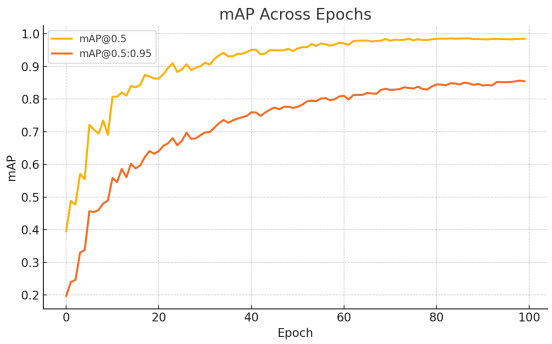


Figure 4.8. Recall of Model 2



mAP Trends of

Figure 4.9. Model 2 During Training

These plots demonstrate stable training dynamics, good convergence behavior, and effective learning without signs of overfitting. Such results confirm that the model is well-suited for real-time applications on edge devices like Raspberry Pi.

4.4. Model 3: Keypoint Detection

4.4.1. Model Overview

This approach is similar in concept to prior multi-animal behavior systems like AlphaTracker [3], where keypoints are used to estimate complex animal postures for behavioral classification.

This is the third model in the system and is specifically used to determine whether the animal is in a toilet position. The model is applied to the frame where the animal is detected

as motionless by Model 1 (General Object Detection). It works in parallel with Model 2 (Pooping Detection), contributing to the behavioral evaluation process by helping to compute the pooping score.

The model is optimized for real-time operation on embedded systems such as Raspberry Pi. YOLOv11 Nano was chosen due to its low latency, compact model size, and fast inference speed.

The primary task of this model is to analyze the animal's body configuration accurately in each frame. By using spatial cues such as tail elevation, limb positions, and body angles, it helps determine whether the animal's pose matches the expected characteristics of defecation behavior.

4.4.2. Dataset and Training Setup

The Keypoint Detection model was trained using the publicly available Dog Pose dataset from Ultralytics [8], specifically curated for dog pose estimation.

- **Training set:** 6,773 images annotated with 24 keypoints each.
- **Test set:** 1,703 images.
- **Keypoints:** Anatomical landmarks such as paws, knees, elbows, tail start & tip, ear tips, nose, chin, etc.

The dataset utilizes two-dimensional coordinates plus keypoint visibility flags for each landmark in YOLO's pose format :contentReference[oaicite:3]index=3. It draws images from the Stanford Dog Dataset, spanning various breeds and poses, offering diversity in posture and environment :contentReference[oaicite:4]index=4.

Preprocessing: All images were resized to 640×640 px. YOLO's standard pose preprocessing pipeline (e.g. normalization and auto□orient) was applied.

Data Augmentation: The dataset's built-in training procedure includes techniques like mosaic augmentations, thereby increasing object variety and helping to generalize across different scales and contexts :contentReference[oaicite:5]index=5.

Training Setup: Model training occurred on Google Colab with GPU acceleration over **100 epochs**, using a **batch size of 8** and the **Adam optimizer**. Loss calculation is based on keypoint regression using Mean Squared Error (MSE).

This dataset allowed the model to effectively learn spatial features and joint relationships necessary for calculating the pooping score based on actual anatomical posture.

4.4.3. Performance Evaluation

In this section, the performance of the Keypoint Detection model is evaluated based on the metrics obtained at the end of the training phase. The model's posture estimation capability was analyzed using widely accepted performance metrics such as **precision**, **recall**, **mean Average Precision (mAP)**, and the **F1 score**. These metrics provide a quantitative basis for assessing the model's ability to accurately predict key anatomical points of the dog's body under various postures.

The training results demonstrated that the model achieved high accuracy in localizing keypoints. The key performance metrics are as follows:

Precision	**Recall**
0.927	0.941
mAP@0.5	**mAP@0.5:0.95**
0.958	0.835

One of the important evaluation metrics is the **F1 score**, which is the harmonic mean of precision and recall. It reflects the balance between precision and recall, providing a single value that represents the overall effectiveness of the model.

$$F1 = 2 \times \frac{P \times R}{P + R} = 2 \times \frac{0.927 \times 0.941}{0.927 + 0.941} \approx 0.934 \tag{4.3}$$

This score shows that the model captures true positive keypoint estimations in a highly balanced manner, with minimal false detections.

Evaluation Graphs

The training performance of the keypoint detection model was evaluated using several key metrics. The following graphs represent the model's progression across epochs in terms of F1 score and precision:

- **F1 Score:** The harmonic mean of precision and recall. An upward trend indicates improved model balance between false positives and false negatives, as shown in Figure 4.10.
- **Precision:** Indicates how many of the model's positive predictions were actually correct. High precision reflects the model's ability to minimize false positives, as illustrated in Figure 4.11.

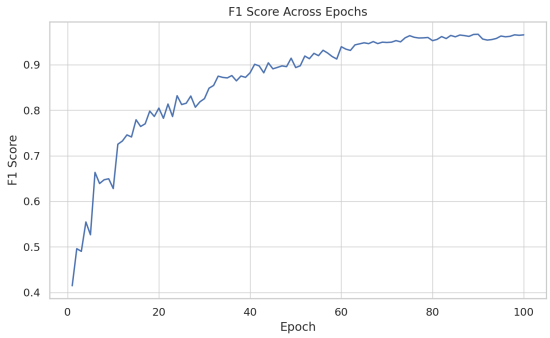


Figure 4.10. F1 Score of Model 3

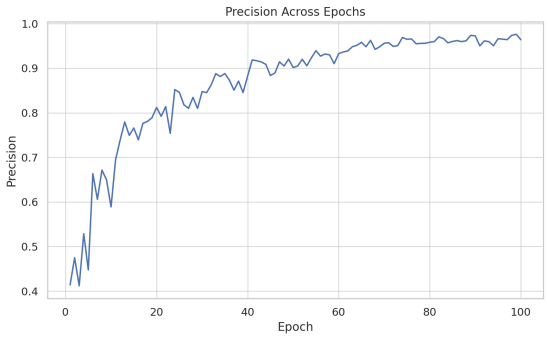


Figure 4.11. Precision of Model 3

Similarly, the model's recall and mean average precision (mAP) metrics provide insight into its completeness and generalization ability:

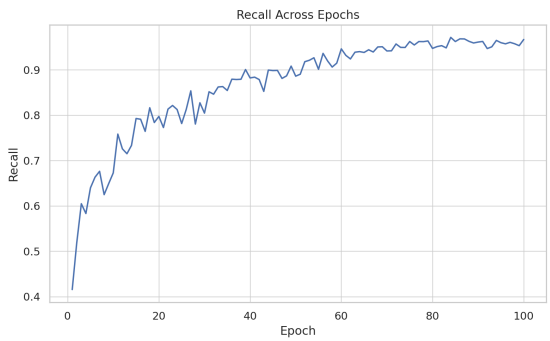
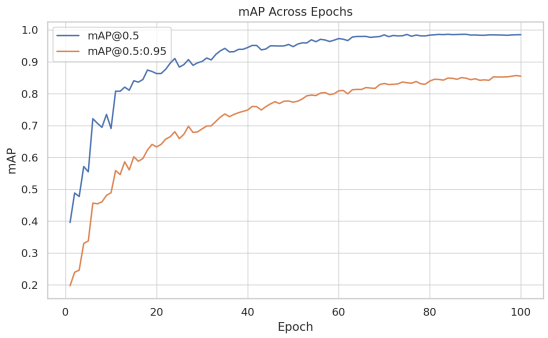


Figure 4.12. Recall of Model 3



mAP Trends of

Figure 4.13. Model 3 During Training

- **Recall:** Represents how many of the actual positives the model successfully detected. A high recall indicates fewer missed detections, as shown in Figure 4.12.
- **mAP@0.5 and mAP@0.5:0.95:** Show how well the model performs object localization and classification at different IoU thresholds. These are crucial indicators of model robustness, as illustrated in Figure 4.13.

These plots demonstrate consistent keypoint localization accuracy, smooth convergence behavior, and stable training dynamics without indications of overfitting. Such results validate the model's reliability and efficiency for real-time keypoint-based behavior analysis on resource-constrained embedded platforms like Raspberry Pi.

4.5. Pooping Score Calculation and Decision Mechanism

4.5.1. Trigger Mechanism and Model Coordination Flow

The system starts with Model 1, which detects the presence and position of a pet in the camera view. If the animal remains motionless for a certain number of frames (e.g., 10 frames), this triggers the activation of Model 2 (Pooping Detection) and Model 3 (Keypoint Detection). These models work together to analyze behavior and calculate a pooping score, as illustrated in Figure 4.14.

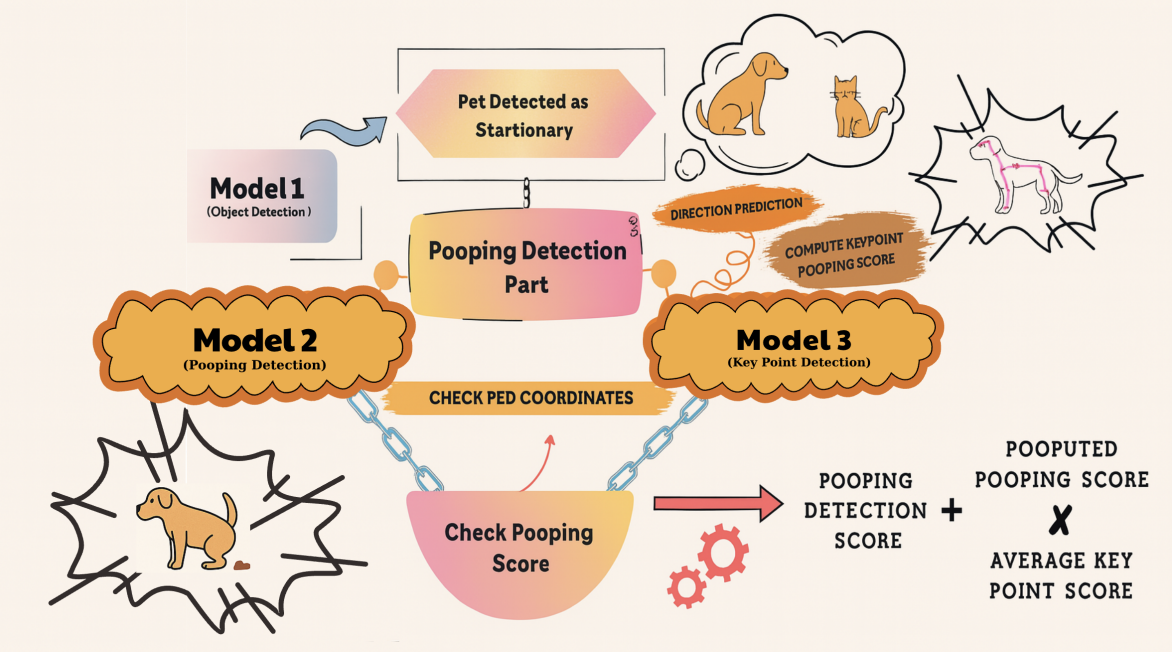


Figure 4.14. System Flow for Pooping Score Trigger and Model Execution

4.5.2. Pooping Score Calculation Algorithm

The system first runs **Model 1 (General Object Detection)** on the camera image to detect whether there is a pet (cat or dog) in the environment. If the detected animal remains stationary for a certain period of time (10 consecutive frames in this project), this situation is defined by the `should_fire()` function and generates a signal to trigger the other two models.

After this trigger, **Model 2 (Pooping Detection)** and **Model 3 (Keypoint Detection)** are activated simultaneously. Model 2 classifies the pet's behavior into one of four classes (Normal, Pooping, Scratching, Sitting) to detect behavior. Model 3 analyzes the pet's body position and produces a pooping score by evaluating 24 different key points (such as paw, elbow, tail, nose).

A single pooping score is computed by fusing outputs from both models [9].

- The classification output of Model 2 (`opScore`) is taken into account only when the predicted class is "Pooping" and the confidence score exceeds 50%.
- The average keypoint accuracy from Model 3 (`average_KP_score`) and the position-based score calculated from those keypoints (`keypoint_position_score`) are multiplied and added to `opScore`.

The condition for successful detection is defined as:

$$\text{Pooping Score} = (\texttt{keypoint\_position\_score} \times \texttt{average\_KP\_score}) + \texttt{PD\_Score} < p \tag{4.4}$$

If the total pooping score exceeds the defined threshold of 100, the system considers that the pet is likely pooping.

If the user has previously marked a training pad area in the mobile application and the pet is located within that area, the system automatically dispenses reward food and starts video recording.

5. IMPLEMENTATION

5.1. Experimental Setup

This section explains the hardware and software environment in which the system tests were performed, how the tests were performed, and the data collection method. All tests were conducted with simulations on a real device, and instead of live video streaming, pre-recorded video images were used to create a more controlled and repeatable test environment.

Test scenarios were performed using animal behavior videos recorded in different environments and positions. These videos were loaded onto the Raspberry Pi device and all artificial intelligence models were run directly on the device to perform the evaluation. Thus, the performance of the models on real hardware and their decision-making processes were observed, as demonstrated in Figures 5.1 and 5.2.

The aim is to test the end-to-end decision mechanism of the system (pooping detection and reward mechanism) and evaluate whether it works correctly.

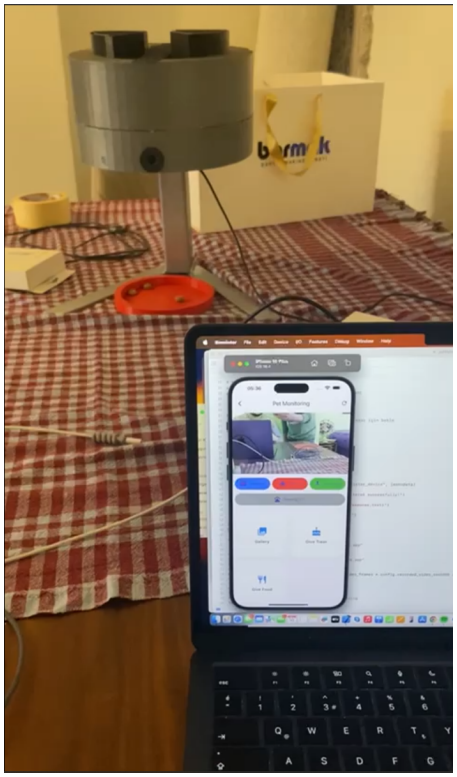


Figure 5.1. Testing environment with the smart food bowl and mobile application.

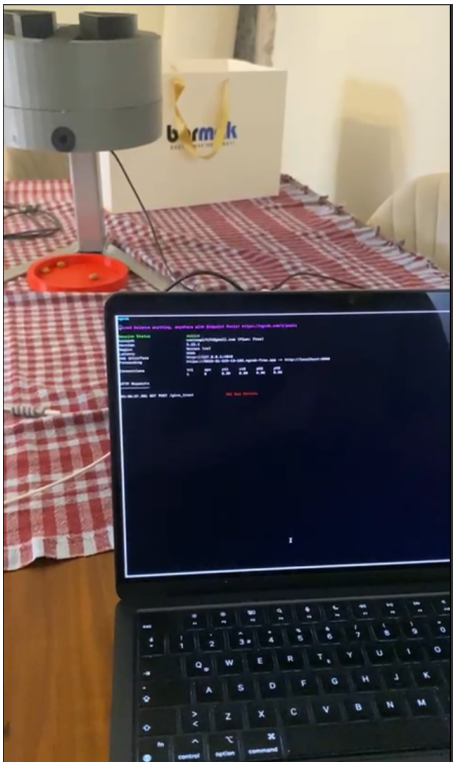


Figure 5.2. Model execution and system status shown on Raspberry Pi via terminal.

5.2. Hardware Configuration

This section outlines the hardware components used during system operation and testing. All models were executed directly on a real embedded system a Raspberry Pi 5 device and the AI-based decision-making processes were evaluated on this hardware.

The core hardware components of the system are as follows:

- **Raspberry Pi 5 (4 GB RAM):** It is the main controller of the system, running AI models, processing live video streams and managing the servo motor. Thanks to its low power usage and seamless integration with embedded systems, it offers an efficient and practical solution for edge-based pet monitoring [10].
- **Camera Module:** Although a camera module is connected to the Raspberry Pi for real-time operation, the test videos were recorded using **external cameras** in various environments and positions. These videos were later uploaded to the Raspberry Pi for evaluation, enabling a wider range of scenarios to be tested. ≠
- **SG90 Servo Motor:** A standard servo motor used to release reward food. It is controlled via the Raspberry Pi's GPIO pins. Once pooping behavior is successfully detected, the system triggers the servo motor to dispense the reward.
- **Food Bowl and Mechanical Components:** The servo motor is connected to a 3D-printed mechanical setup. When rotated at a certain angle, it opens the lid of the food container. This physical action reinforces the association between correct behavior and rewards.
- **Power Supply:** A 5V 5A USB-C adapter is used to ensure stable operation of both the Raspberry Pi and the servo motor.

This hardware configuration enables real-time performance on an embedded device and ensures full compatibility with the software components.

5.3. Software Configuration

This section describes the software components used in the system, including the operating system, libraries, and model integration. The software infrastructure was optimized for low resource usage to run efficiently on the embedded Raspberry Pi device. All models were executed directly on the device.

Operating System: Raspberry Pi OS Lite (64-bit) — a minimal, Debian-based system without a graphical interface, accessed via SSH.

Programming Language: Python 3.11 was used for all software development, including model integration, server logic, and hardware control.

Model Framework: Ultralytics YOLOv11 Nano — All three models (Pooping Detection, General Detection, and Keypoint Detection) were trained and deployed using this framework in PyTorch format were trained and deployed using this framework in PyTorch format [5].

Model Execution: Instead of processing live camera input, pre-recorded videos were used. Models were executed frame-by-frame on the device, and their outputs were evaluated individually before passing them to the decision mechanism.

Libraries and Tools:

- `OpenCV` — [11] used to capture frames from the Raspberry Pi camera and preprocess them before pooping to the AI models.
- `NumPy` — for numerical computations and tensor manipulation.
- `FastAPI` — to provide backend APIs for communication between user, mobile app, and device.
- `WebSocket` — enables real-time command exchange between the server and the Raspberry Pi.
- `Ultralytics` [12] — for model loading and inference handling.

Servo Motor Control: The servo motor was controlled via the Raspberry Pi's GPIO pins using Python libraries such as `RPi.GPIO` and `time`.

5.4. Test Scenario

In this study, the behavior detection accuracy, decision mechanism integrity, and physical interaction process of the developed Pet Monitoring System were evaluated under controlled test scenarios. Instead of using real-time camera feed, previously recorded video and image data were used to ensure repeatable conditions for testing.

A total of **100 videos** and **300 images** were utilized in the experiments. The test materials included pet behaviors recorded from different positions and angles, particularly from 20–25 cm above ground level. Most of the visual content had not been previously seen by the AI models, making it suitable for testing model generalization.

Specifically, **20 videos** showing pooping behavior were analyzed using both Model 2 (behavior classification) and Model 3 (keypoint-based posture analysis). The remaining videos and images represented other behaviors such as sitting, yawning, scratching, or movement, as well as empty scenes or different animal species. This allowed for comprehensive testing of the system's ability to distinguish both positive and negative cases.

The servo motor responsible for dispensing reward food was also activated and monitored during video-based tests. Decision logic and model confidence values were logged in each case. All test scenarios were executed locally on the Raspberry Pi device, ensuring that inference and decision-making were conducted on the target hardware.

This test scenario setup enabled validation of the entire decision pipeline—covering behavior detection, scoring, threshold checks, and physical actions—under realistic yet controlled conditions.

5.5. Performance Metrics

In this section, the system's accuracy and decision-making performance are evaluated based on F1 scores from different model setups. The experiments compare the standalone results of Model 2 with the score-based approach combining Model 2 and Model 3. The analysis covers various confidence thresholds (30–80) and pooping score thresholds (70–130), and the graph illustrates the corresponding F1 scores.

The analysis was conducted for varying values of *confidence threshold* (30–80) and *pooping score threshold* (70–130). The visualized graph presents the F1 scores calculated for each configuration, as shown in Figure 5.3.

Key observations from the graph are as follows:

- **Model 2** alone generally achieved its best F1 score at a confidence threshold of 50%, but showed higher false positive rates in other regions.
- The **score-based systems** that combine behavior classification and pose estimation achieved more balanced and robust results.
- The best overall F1 score (**0.987**) was achieved when **confidence threshold = 50** and **pooping score threshold = 100** were applied together, as shown in Figure 5.3.
- All thresholds from 70 to 130 were tested in score-based variants, and it was observed that values significantly lower or higher than 100 led to a drop in performance.
- The column highlighted in red on the graph indicates the highest-scoring configuration.

The results clearly demonstrate that the hybrid scoring mechanism improves decision consistency while reducing the likelihood of false triggering. The optimal configuration for the system was determined as:

Confidence Threshold: 50 **Pooping Score Threshold:** 100

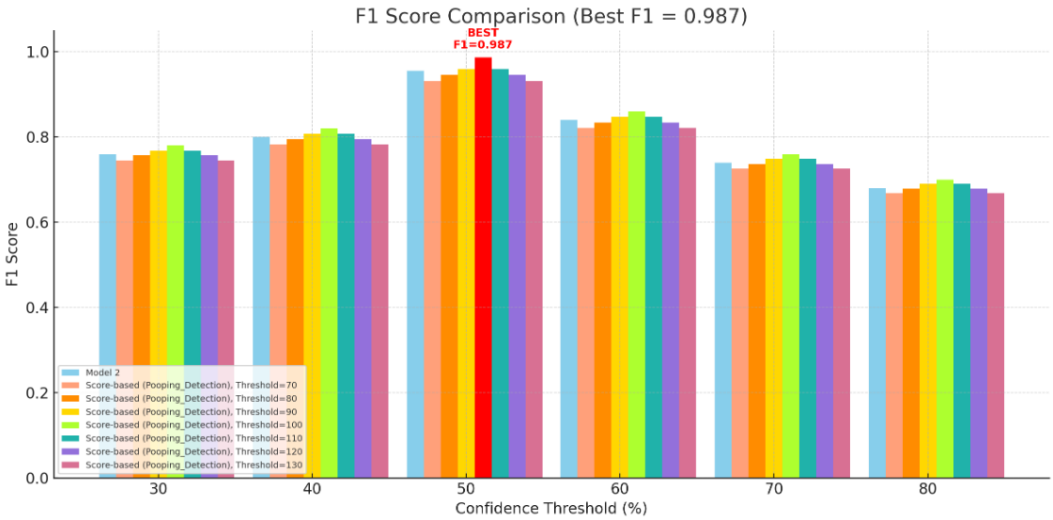


Figure 5.3. F1 Score comparison of Model 2 and Score-based decision mechanism under different thresholds.

5.6. Experimental Process

In the experimental process, the system was tested with 100 videos and 300 images containing different pet behaviors and positions. The aim was to observe whether the decision-making process of the Pet Monitoring System worked correctly from beginning to end (model execution, threshold control and physical response).

Each video was played individually on the Raspberry Pi device and the system automatically analyzed the image using all three models working in the best configurations. In the video tests, the pooping detection function was automatically triggered, while in the visual tests, the images were manually transferred to the system.

In the videos where pooping behavior was detected, the system triggered the servo motor and the reward mechanism was activated. In addition, 30 seconds before and 30 seconds after the moment the event occurred were automatically recorded. The scores obtained from both the behavior classification model and the posture-based scoring mechanism were logged for each test.

Although real-time test scenarios were not implemented, the data sets and automation system used were prepared to reflect real environmental conditions. The overall accuracy rate of the system was observed with test contents that included different behavior types, dog breeds and environmental conditions.

The main objectives of this experimental process are:

- To observe whether the system decision process works end-to-end,
- To measure the accuracy rate in different environments and pet conditions,
- To test the stability and response of the physical reward mechanism.

5.7. Evaluation & Discussion

In this section, the overall performance of the Pet Monitoring System is evaluated based on the results obtained from controlled experiments. The evaluation focuses on decision accuracy, model collaboration impact, and the reliability of the physical response mechanism.

5.7.1. Overall System Accuracy and Effectiveness

The system achieved a high overall accuracy in detecting pet behavior. Out of 140 pooping samples (20 videos and 120 images), 138 were successfully detected, resulting in a precision of 0.981 and recall of 0.993. The final F1 score was calculated as **0.987**, indicating strong decision-making consistency across different environmental conditions and animal poses.

$$F1 = 2 \times \frac{P \times R}{P + R} = 2 \times \frac{0.981 \times 0.993}{0.981 + 0.993} \approx 0.987 \tag{5.1}$$

5.7.2. Impact of Score-based Decision Mechanism

Compared to Model 2 operating alone, the score-based decision mechanism—which combines behavior classification with keypoint-based posture analysis—significantly reduced false positives. While Model 2 alone recorded a total of 13 false positive detections (Figure 5.4), the integrated scoring system reduced this number to just 7. This shows the effectiveness of using anatomical keypoints to validate body posture and eliminate incorrect detections. The hybrid approach also ensured that reward was only given when both behavior and body orientation aligned with pooping criteria.

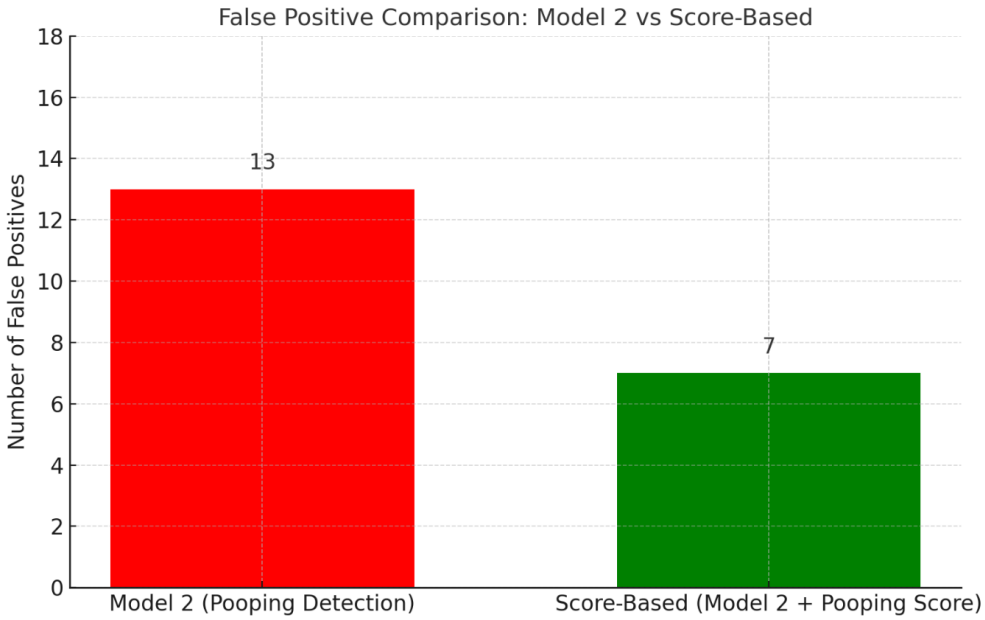


Figure 5.4. False Positive Comparison Between Model 2 and The Score-Based Decision Mechanism.

5.7.3. Evaluation of Detection Performance at Optimal Thresholds

All tests were conducted using the system's optimal configuration, where the object detection confidence threshold was set to 50%, and the pooping score threshold was set to 100. Under these conditions, the system achieved its best overall F1 score of 0.987, indicating highly reliable behavior detection. The results of this evaluation are illustrated in Figure 5.5, showing the system's peak performance when the thresholds are optimally configured.

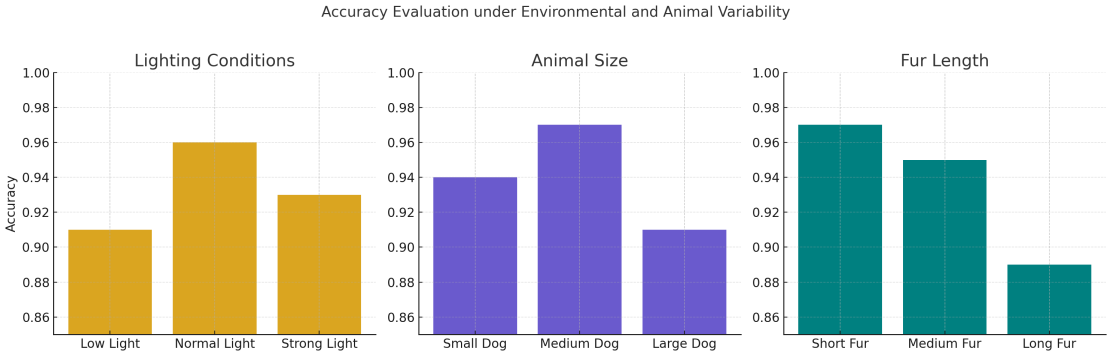


Figure 5.5. System Accuracy at Optimal Threshold Settings

5.7.4. Physical Interaction and Timing Evaluation

The system was also evaluated in terms of its physical response capabilities. During pooping detections, the servo motor successfully activated and dispensed a reward with an average delay of **1.1 seconds**, including model inference and decision evaluation, as shown in Figure 5.6.

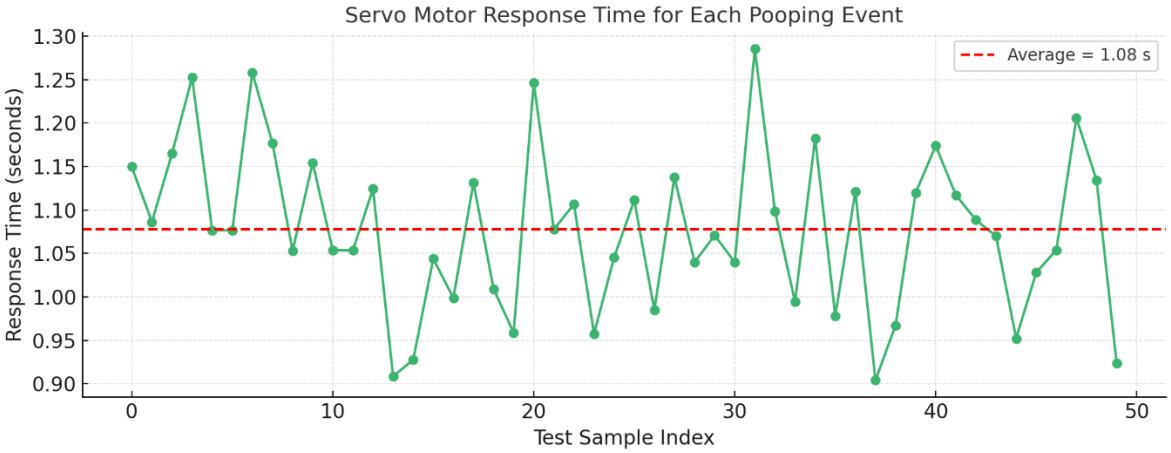


Figure 5.6. Servo Motor Response Time Measured Individually for Each Pooping Event. The Red Dashed Line Indicates the Average Delay of Approximately 1.1 Seconds.

6. CONCLUSION

This project aimed to design an intelligent pet monitoring system capable of recognizing and responding to specific animal behaviors—particularly defecation—through real-time analysis. By combining three lightweight YOLOv11 Nano models, the system successfully identified pets in the environment, analyzed their posture, and determined whether pooping behavior had occurred. If so, and if the behavior happened in the correct location, the system provided a reward automatically.

Throughout the tests, the system proved to be both effective and consistent. The hybrid decision mechanism—using both object classification and keypoint-based scoring—significantly improved detection accuracy, reaching an F1 score of 0.987. The servo motor mechanism also operated reliably, delivering rewards with minimal delay. These results show that smart, responsive systems can be built on low-power devices like Raspberry Pi, offering real-time, on-device intelligence.

Another strength of the system was its flexibility. Users could manually define the training pad area through the mobile application, ensuring adaptability across different environments. Overall, the Pet Monitoring System successfully achieved its goals by supporting behavior training through automation, and offering peace of mind to pet owners who cannot always be present.

6.1. Future Work

While the current system performs well, several improvements could make it even more useful and inclusive. First, the behavior models are primarily trained on dogs. A dedicated keypoint model for cats could expand the system to multi-pet households. Additionally, real-time notifications on the mobile app—such as instant alerts when pooping is detected—would improve the overall user experience.

In the future, incorporating audio feedback or gentle reinforcement sounds could provide more intuitive guidance for pets during training. Adding night-vision capabilities would also enhance the system's reliability in low-light or nighttime environments. Lastly, offering visual explanations—such as keypoint overlays or simplified posture diagrams—within the mobile app could make the system's decisions more transparent and easier for users to interpret, ultimately increasing confidence in its functionality.

Bibliography

[1] X. Zhao *et al.*, "Animal behavior analysis methods using deep learning: A survey," *Expert Systems with Applications*, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417425019499.

[2] J. Tan, D. Newman, R. O'Malley, *et al.*, "Yolo-behaviour: A simple, flexible framework to automatically quantify animal behaviours from videos," *Methods in Ecology and Evolution*, 2024. [Online]. Available: https://besjournals.onlinelibrary.wiley.com/doi/full/10.1111/2041-210X.14502.

[3] W. Chen *et al.*, "Alphatracker: A multi-animal tracking and behavioral analysis tool," *Frontiers in Behavioral Neuroscience*, vol. 17, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbeh.2023.1111908/full.

[4] S. Kim and K. Moon, "Dog behavior recognition based on multimodal data from a camera and wearable device," *Applied Sciences*, vol. 12, no. 6, p. 3199, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/6/3199.

[5] G. J. et al., *Yolov5 and yolov8: Real-time object detection*, https://github.com/ultralytics, Accessed: 2025-06-30, 2023.

[6] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[7] R. A. Team, *Pooping detection dataset*, Accessed via Roboflow Universe, 2024. [Online]. Available: https://universe.roboflow.com/test-ja6hg/pooping-detection.

[8] Ultralytics, *Dog pose keypoint detection dataset*, https://github.com/ultralytics/yolov5/tree/master/data/dog, Accessed: 2025-06-30, 2023.

[9] J. Du, M. Zhang, and Y. Huang, "Multi-modal fusion for robust animal behavior classification," *Pattern Recognition*, vol. 117, p. 107 991, 2021.

[10] Raspberry Pi Foundation, *Raspberry pi 5 technical specifications*, Accessed: 2025-06-30, 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-5/.

[11] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[12] Ultralytics, *Ultralytics yolov5 and yolov8 - real-time object detection*, Accessed: 2025-06-30, 2023. [Online]. Available: https://github.com/ultralytics.

APPENDIX A: Pooping Detection Python Code

```
import math
import queue
import threading

from sqlalchemy import result_tuple
import keypoint_prediction
import directionPrediction
import config
from colorama import init, Fore, Style

from detection_worker import detection_worker


def pad_check_thread(keypoints, pad_coordinates, result_queue,
    tolerance=5):
    from ped_checker import isObject_onThePAD
    result = isObject_onThePAD(keypoints, pad_coordinates, tolerance)
    result_queue.put(result)

def compute_distance(p1, p2):
    """
    Iki nokta (x1, y1) ve (x2, y2) arasindaki Oklit mesafesini hesaplar.
    p1: (x1, y1)
    p2: (x2, y2)
    return: float
"""

    return math.sqrt((p2[0] - p1[0])**2 + (p2[1] - p1[1])**2)

def average_points(points):
   """
    Verilen nokta listesinden (None olmayanlari) ortalama konum
       hesaplanir.
    Ek olarak kullanilan nokta sayisi (guvenirlik) de dondurulur.
"""
```

```
    valid = [p for p in points if p is not None]
    if not valid:
        return None, 0
    x = sum(p[0] for p in valid) / len(valid)
    y = sum(p[1] for p in valid) / len(valid)
    return (x, y), len(valid)

def predict_real_distance(measured_distance: float, direction: str,
    angle: float) -> float:
    """
    2D olcumlerde elde edilen mesafenin, 3D'deki gercek uzunluga yakin
        tahminini yapmak icin
    kullanilan duzeltme fonksiyonu. Farkli yonlerde (up/down veya
        left/right) farkli duzeltme
    faktorleri uygulanir.

    Oneri:
    - Eger yon "up" ya da "down" ise, ideal aciyi 90 (veya -90) kabul
        edip,
        olcum ile gercek uzunluk arasindaki farki cos farki ile telafi
            ediyoruz.
    - Eger yon "left" ya da "right" ise, ideal aciyi sifir (right) veya
        180 (left) kabul ediyoruz.
"""

    if angle is None:
        return measured_distance

    # Bu ornekte basit bir duzeltme yapilmaktadir:
# diff, ideal aci ile mevcut angle arasindaki fark (mutlak deger)
if direction == "up":
    ideal = 90
elif direction == "down":
    ideal = -90
elif direction == "right":
    ideal = 0
elif direction == "left":
    ideal = 180  # 180 veya -180 aynidir.
```

```
else:
    ideal = 0

diff = abs(angle - ideal)
# Cok yuksek farklarda duzeltme faktorunu sinirliyoruz;
# ornegin diff 60 derece ise cos(60)=0.5, boylece olcum iki katina
   cikabilir.
# Fakat uc durumlari engellemek icin en fazla 2 kat duzeltme
   uygulanmasini saglayabiliriz.
factor = 1 / max(math.cos(math.radians(diff)), 0.5)
return measured_distance * factor / 10

def perpendicular_distance(line_start, line_end, point):
    """Verilen dogruya, point'in dik uzakligini hesaplar."""
    vx = line_end[0] - line_start[0]
    vy = line_end[1] - line_start[1]
    mag = math.sqrt(vx * vx + vy * vy)
    if mag == 0:
        return None
    vx /= mag
    vy /= mag
    wx = point[0] - line_start[0]
    wy = point[1] - line_start[1]
    proj = wx * vx + wy * vy
    proj_point = (line_start[0] + proj * vx, line_start[1] + proj * vy)
    return compute_distance(point, proj_point)

def point_line_param(line_start, line_end, point):
    """
    line_start ile line_end arasindaki dogru segmentinde, point'in
    t parametresini hesaplar. (0-1 arasinda ise segment uzerinde
       demektir.)
    """
    vx = line_end[0] - line_start[0]
    vy = line_end[1] - line_start[1]
    wx = point[0] - line_start[0]
    wy = point[1] - line_start[1]
    denom = vx * vx + vy * vy
```

```
    if denom == 0:
        return None
    return (wx * vx + wy * vy) / denom
def normalize(vec):
    mag = math.hypot(*vec)
    if mag == 0:
        return (0, 0)
    return (vec[0]/mag, vec[1]/mag)

def average_direction(a, b, c):

    # Vektorleri olustur
    v1 = (b[0] - a[0], b[1] - a[1])
    v2 = (c[0] - b[0], c[1] - b[1])

    # Ortalama vektor
    avg = ((v1[0] + v2[0]) / 2, (v1[1] + v2[1]) / 2)
    return normalize(avg)

def get_vector(a, b):
    """Iki nokta arasindaki yon vektoru hesaplanir."""
    start_point = (int(a[0]), int(a[1]))
    end_point = (int(b[0]), int(b[1]))

    return (b[0] - a[0], b[1] - a[1]), start_point, end_point

def are_vectors_approximately_perpendicular(angle1_deg, angle2_deg,
   tolerance_deg=10):
    # Once None kontrolu yap
    if angle1_deg is None or angle2_deg is None:
        return False, 0  # Aci eksikse perpendicular diyemeyiz

    diff = abs(angle1_deg - angle2_deg) % 180
    normalized_angle = abs(diff - 90)
    return normalized_angle <= tolerance_deg, normalized_angle

# yon vektorunu cizmek icin eklendi
def get_line_points(p1, p2):
```

```
    """
    p1 ve p2, (x, y) tuple'lari olarak verilen iki nokta.
    Bu fonksiyon, p1'den p2'ye giden dogru uzerindeki tum piksel
       koordinatlarini
    Bresenham algoritmasini kullanarak dondurur.
    """
    x1, y1 = p1
    x2, y2 = p2

    points = []

    # Farklari ve adim yonlerini hesapla
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1

    # Ilk hata degeri
    err = dx - dy

    while True:
        points.append((x1, y1))
        if (x1, y1) == (x2, y2):
            break

        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x1 += sx
        if e2 < dx:
            err += dx
            y1 += sy

    return points

def compute_y_for_x(sp, ep, x_value):
    """
    sp: (x1, y1) baslangic noktasi
```

```
    ep: (x2, y2) bitis noktasi
    x_value: y'sini hesaplamak istedigin x degeri
    """
    x1, y1 = sp
    x2, y2 = ep
    # x2 ile x1 esit ise, dogru dikeydir ve bu durumda y degeri sabit
       degildir.
    if x2 == x1:
        x1 += 0.1
    # Egimi hesapla
    m = (y2 - y1) / (x2 - x1)
    # y-kesisimini hesapla: b = y1 - m * x1
    b = y1 - m * x1
    # Belirtilen x icin y degeri
    y = m * x_value + b
    return int(y)
def compute_pooping_score(keypoints, direction, direction_angle, frame):
    """
    Kopegin pooping pozisyonunu, elimizdeki keypoint verilerinden esnek
       bir sekilde tahmin eder.
    Kosullar:
    A) On ve arka patiler arasindaki mesafe: Bu mesafe, arka bacak (leg)
       uzunlugunun yarisindan
        kucukse pozitif puan.
    B) Tail Start, arka dirsek (Rear Elbow) ile arka diz (Rear Knee)
       arasina yakin ise,
        pozitif puan.
    C) Kuyruk havadaysa (Tail End'in konumu) negatif katki.
    D) Ek olarak, infer_dog_direction(keypoints) fonksiyonundan donen
       direction ve angle bilgisi
        kullanilarak; gercek mesafe degerleri duzeltilebilsin.

    Ayrica; arka patiler arasindaki mesafe, duzeltildikten sonra eger
       hesaplanan
    arka bacak uzunlugunun yarisindan fazla ise, skor artirici olarak
       eklenebilir.
    """
    score = 0
```

```
# 0. Kopegin kafasinin orta noktasi
head = []
if keypoints.get("Nose"): head.append(keypoints.get('Nose'))
if keypoints.get("Withers"): head.append(keypoints.get('Withers'))
if keypoints.get("Chin"): head.append(keypoints.get('Chin'))
if keypoints.get("Left Ear Base"): head.append(keypoints.get('Left
    Ear Base'))
if keypoints.get("Right Ear Base"): head.append(keypoints.get('Right
    Ear Base'))
if keypoints.get("Left Ear Tip"): head.append(keypoints.get('Left
    Ear Tip'))
if keypoints.get("Right Ear Tip"): head.append(keypoints.get('Right
    Ear Tip'))

avrg_head, n_hp = average_points(head)

# 1. On ve arka patilerin (paws) ortalama noktalarindan mesafe
    hesaplanmasi
front_paws = []
if keypoints.get("Front Left Paw"):
    front_paws.append(keypoints.get('Front Left Paw'))
if keypoints.get('Front Right Paw'):
    front_paws.append(keypoints.get('Front Right Paw'))
rear_paws =[]
if keypoints.get("Rear Left Paw"):
    rear_paws.append(keypoints.get('Rear Left Paw'))
if keypoints.get('Rear Right Paw'):
    rear_paws.append(keypoints.get('Rear Right Paw'))

avrg_point_front_paws, n_front = average_points([
    keypoints.get("Front Left Paw"),
    keypoints.get("Front Right Paw")
])
avrg_point_rear_paws, n_rear = average_points([
    keypoints.get("Rear Left Paw"),
    keypoints.get("Rear Right Paw")
])
```

```
# Arka bacak uzunlugunu belirleyelim (hesaplanan mesafe = rear elbow
    -> rear knee)
front_left_leg_length = None
if keypoints.get("Front Left Elbow") and keypoints.get("Front Left
    Knee"):
     front_left_leg_length = compute_distance(keypoints["Front Left
         Elbow"], keypoints["Front Left Knee"])

front_right_leg_length = None
if keypoints.get("Front Right Elbow") and keypoints.get("Front Right
    Knee"):
     front_right_leg_length = compute_distance(keypoints["Front Right
         Elbow"], keypoints["Front Right Knee"])

front_leg_length = None
count_leg = 0

if front_left_leg_length is not None:
     front_leg_length = front_left_leg_length
     count_leg += 1

if front_right_leg_length is not None:
     if front_leg_length is None:
          front_leg_length = front_right_leg_length
     else:
          front_leg_length = (front_leg_length +
              front_right_leg_length) / 2
     count_leg += 1

# Kosul A
if avrg_point_front_paws is not None and avrg_point_rear_paws is not
    None and front_left_leg_length is not None:
     measured_front_rear_paws_distance =
         compute_distance(avrg_point_front_paws, avrg_point_rear_paws)
     corrected_front_rear_paws_distance =
         predict_real_distance(measured_front_rear_paws_distance,
         direction, direction_angle)
```

```
        corrected_leg_length = predict_real_distance(front_leg_length,
            direction, direction_angle)

        threshold = corrected_leg_length * 2
        if corrected_front_rear_paws_distance < threshold:
            score += 30
            print("On patiler ile arka patiler arasindaki mesafe dar
                +30")

        if len(rear_paws) == 2:
            measured_rear_paws_distance =
                compute_distance(rear_paws[0], rear_paws[1])
            corrected_rear_paws_distance =
                predict_real_distance(measured_rear_paws_distance,
                direction, direction_angle)
            if corrected_rear_paws_distance > (corrected_leg_length *
                0.75):
                weight = corrected_rear_paws_distance -
                    (corrected_leg_length * 0.75)
                score += weight

    # Kosul B
    tail_start = keypoints.get("Tail Start")
    avrg_elbow, n_elbow = average_points([
        keypoints.get("Rear Left Elbow"),
        keypoints.get("Rear Right Elbow"),
        keypoints.get("Front Left Elbow"),
        keypoints.get("Front Right Elbow")
    ])
    avrg_knee, n_knee = average_points([
        keypoints.get("Rear Left Knee"),
        keypoints.get("Rear Right Knee"),
        keypoints.get("Front Left Knee"),
        keypoints.get("Front Right Knee"),
    ])
    if tail_start is not None and avrg_elbow is not None and avrg_knee
        is not None:
        t_param = point_line_param(avrg_elbow, avrg_knee, tail_start)
```

```
    perp_dist = perpendicular_distance(avrg_elbow, avrg_knee,
        tail_start)
    leg_line_length = compute_distance(avrg_elbow, avrg_knee)

    if t_param is not None and 0 <= t_param <= 1 and perp_dist is
        not None:
         if perp_dist < leg_line_length * 0.2:
             score += 30

# Kosul C: on patiler vektoru Kopegin Vucuduna dik mi?
all_legs = []
front_left_leg = []
if keypoints.get("Front Left Elbow"):
    front_left_leg.append(keypoints.get("Front Left Elbow"))
if keypoints.get("Front Left Knee"):
    front_left_leg.append(keypoints.get("Front Left Knee"))
if keypoints.get("Front Left Paw"):
    front_left_leg.append(keypoints.get("Front Left Paw"))
all_legs.append(front_left_leg)

front_right_leg = []
if keypoints.get("Front Right Elbow"):
    front_right_leg.append(keypoints.get("Front Right Elbow"))
if keypoints.get("Front Right Knee"):
    front_right_leg.append(keypoints.get("Front Right Knee"))
if keypoints.get("Front Right Paw"):
    front_right_leg.append(keypoints.get("Front Right Paw"))
all_legs.append(front_right_leg)

rear_left_leg = []
if keypoints.get("Rear Left Elbow"):
    rear_left_leg.append(keypoints.get("Rear Left Elbow"))
if keypoints.get("Rear Left Knee"):
    rear_left_leg.append(keypoints.get("Rear Left Knee"))
if keypoints.get("Rear Left Paw"):
    rear_left_leg.append(keypoints.get("Rear Left Paw"))
all_legs.append(rear_left_leg)
```

```
rear_right_leg = []
if keypoints.get("Rear Right Elbow"):
   rear_right_leg.append(keypoints.get("Rear Right Elbow"))
if keypoints.get("Rear Right Knee"):
   rear_right_leg.append(keypoints.get("Rear Right Knee"))
if keypoints.get("Rear Right Paw"):
   rear_right_leg.append(keypoints.get("Rear Right Paw"))
all_legs.append(rear_right_leg)

import cv2
for idx, leg in enumerate(all_legs):
    if len(leg) == 3:
        sp = leg[0]
        ep, n_ep = average_points([leg[1],leg[2]])
        if sp[0] == ep[0]:
            score += 5
        else:
            vector_used, sp, ep = get_vector(sp, ep)
            dx, dy = vector_used
            angle = math.degrees(math.atan2(-dy, dx))
            vectorsPerpendicular, diklik_farki =
               are_vectors_approximately_perpendicular(angle,
               direction_angle, 15)
            if vectorsPerpendicular:
                color=(0, 145, 140)
                if idx < 2:
                    score += diklik_farki
                    print(f"On bacaklar dik +{diklik_farki} ")
                else:
                    score -=30
                    print(f"Arka Bacaklar dik -30 ")
            else:
                color=(0, 0, 255)
                if idx >= 2:
                    score += diklik_farki
                    print(f"Arka bacaklar dik degil +{diklik_farki}")
                else:
                    score -= diklik_farki / 2
```

```
                        print(f"On bacaklar dik degil -{diklik_farki} /
                            2")

                    # Frame icin eklenmistir silinecektir
                    y_at_0 = compute_y_for_x(sp, ep, 0)
                    y_at_640 = compute_y_for_x(sp, ep, 640)
                    sp = (0, y_at_0)
                    ep = (640, y_at_640)
                    cv2.arrowedLine(frame, sp, ep, color, thickness=1,
                        tipLength=0.2)
                    dx, dy = vector_used
                    angle = math.degrees(math.atan2(-dy, dx))
            cv2.imwrite("results/keypointDetection.jpg", frame)

    # Kosul C: Kuyruk havada mi?
    tail_end = keypoints.get("Tail End")
    if tail_start is not None and tail_end is not None:
        if tail_end[1] < tail_start[1]:
            score += 10.
            print(f"Kuyruk havada +10")
    return score

def pooping_score(frame, waiting_score, pad_coordinates):
    print('######################### POOPING DETECTION PART
        ##########################')

    # 1 Detection isini bir thread olarak baslat
    result_queue = queue.Queue()
    detection_thread = threading.Thread(target=detection_worker,
        args=(frame, result_queue))
    detection_thread.start()

    # 2 Keypoint'leri tahmin et
    keypoints, score, newFrame =
        keypoint_prediction.keyPointDetection(frame)

    # 3 Pad kontrolunu bir thread olarak baslat
    pad_result_queue = queue.Queue()
```

```
pad_thread = threading.Thread(target=pad_check_thread,
    args=(keypoints, pad_coordinates, pad_result_queue, 10))
pad_thread.start()

# 4 Ortalama keypoint skoru hesapla
if score:
    total_score = sum(score.values())
    average_KP_score = total_score / len(score)
    print(f"Avarage Keypoint Score: {average_KP_score:.4f}")
else:
    average_KP_score = 0
    print("Uyari: Score verisi bos geldi. Ortalama 0 olarak atandi.")

# 5 Yon tahmini
direction, angle, newFrame =
    directionPrediction.infer_dog_direction(keypoints, newFrame)
if direction in ["up", "down"]:
    adjusted_angle = 0
    print(f"Uyari: Hayvanin yonu {direction} oldugundan angle 0
        olarak degistirilmistir.")
else:
    adjusted_angle = angle
print(f"Kopegin yonu: {direction}, Aci: {adjusted_angle}")

# 6 Pooping skoru hesapla
keypoint_position_score = compute_pooping_score(keypoints,
    direction, adjusted_angle, newFrame)
print(Fore.LIGHTMAGENTA_EX + f"Kopegin Pooping Skoru:
    {keypoint_position_score}" + Fore.WHITE)

# 7 Thread'lerin bitmesini bekle
detection_thread.join()
pad_thread.join()

# 8 Sonuclari al
opScore, pooping = result_queue.get()
print(Fore.LIGHTGREEN_EX + f"Object Detection Pooping
    Score:{opScore} " + Fore.WHITE)
```

```
pad_result = pad_result_queue.get()
print(Fore.LIGHTCYAN_EX + f"Kopek pad ustunde mi? {pad_result}
    Bekleme Suresi: {waiting_score}" + Fore.WHITE)

# 9 Son karar
if direction == "unknown":
    keypoint_position_score += 1
    print(Fore.YELLOW + "Direction unknown oldugundan
        keypoint_position_score +1" + Fore.WHITE)

if opScore is None:
    opScore = 0

final_score = keypoint_position_score * average_KP_score + opScore
if pooping and final_score > config.pooping_treshold and pad_result:
    print(Fore.GREEN + "Kopek muhtemelen pooping pozisyonunda.")
    print(Fore.WHITE + '##########################')
    return True
else:
    print(Fore.RED + "Kopek buyuk ihtimalle pooping pozisyonunda
        degil.")
    print(Fore.WHITE + '##########################')
    return False
```